WHAT IS CLAIMED IS:

1. In the operation of an embedded processor complex for controlling the programmability of a network processor, the processor complex including a plurality of protocol processor units (PPUs), each PPU containing at least one core language processor (CLP), each CLP having at least two code threads, each PPU utilizing a plurality of coprocessors useful for executing specific tasks for the PPU, and multiple logical coprocessor interfaces to provide access between each CLP and the coprocessors.

2. In the operation according to claim 1, wherein the coprocessors include dedicated coprocessors that support multiple code threads of each CLP.

3. In the operation according to claim 1, wherein the coprocessors are selected from the group including a tree search coprocessor, a checksum coprocessor, a stringcopy coprocessor, an enqueue coprocessor, a datastore coprocessor, a CAB coprocessor, a counter coprocessor and a policy coprocessor.

4. In the operation according to claim 3 further including a coprocessor execution interface arbiter to determine the priority between multiple threads.

5. In the operation according to claim 3 including a coprocessor data interface arbiter that determines the priority between data threads.

6. In the operation according to claim 3 further including a FIFO buffer between each thread and at least one coprocessor.

7. In the operation according to claim 6, wherein the FIFO buffer is between each thread and the counter coprocessor.

8. In the operation according to claim 6, wherein the FIFO buffer is between each thread and the policy coprocessor.

9. A network processing system including an embedded processor complex for controlling the programmability of a network processor, said complex including a plurality of protocol processor units (PPUs), each PPU containing:
at least one core language processor (CLP), each CLP having at least two code threads;
a plurality of coprocessors for executing specific tasks for the system, and
multiple coprocessor interfaces to access and share the resources of the coprocessors with each CLP.

10. The network processing system of claim 9 wherein the coprocessor interfaces are dedicated to supporting the code threads of each CLP.

11. The network processing system of claim 10 wherein the coprocessors are selected from the group including a tree search coprocessor, checksum coprocessor, stringcopy coprocessor, enqueue coprocessor, datastore coprocessor, CAB coprocessor, counter coprocessor and policy coprocessor.

12. The network processing system of claim 10 further including a FIFO buffer between each thread and at least one of the coprocessors.

13. The network processing system of claim 12 wherein the FIFO buffer is between each thread and the counter coprocessor.

14. The network processing system according to claim 12 wherein the FIFO buffer is between each thread and the policy coprocessor.

15. The network processing system of claim 9 including specific operating

instructions executed by the threads of the CLPs which result in commands to control coprocessor operation, which commands flow through the interface between the CLPs and the coprocessors.

16. The network processing system according to claim 15 wherein the instructions serve to enable conditional execution of specific coprocessor operations.

17. The network processing system according to claim 15 wherein the instructions enable the system to identify long latency events and short latency events according to the expected response time to access data in response to a particular coprocessor command, and to grant full control to another thread when execution of an active thread stalls due to a long latency event, or to grant temporary control to another thread when execution of an active thread stalls due to a short latency event.

18. A method of controlling the execution of instructions within an embedded processor complex which contains a plurality of protocol processor units (PPUs), each protocol processor unit containing at least one core language processor (CLP), each CLP having at least two code threads, comprising the use by each PPU of a plurality of coprocessors for executing specific tasks for the PPUs, and the use of multiple logical coprocessor interfaces to provide access between the coprocessors and each CLP.

19. The method according to claim 18 including the use of dedicated coprocessors that support the multiple code threads of the PPU.

20. The method according to claim 19 wherein one or more of the coprocessors are selected from the group including a tree search coprocessor, checksum coprocessor, stringcopy coprocessor, enqueue coprocessor, datastore coprocessor, CAB coprocessor, counter coprocessor and policy coprocessor.

21. The method according to claim 20 wherein a coprocessor execution interface arbiter serves to determine the priority between execution threads.

22. The method according to claim 20 wherein a coprocessor data interface arbiter serves to determine the priority between data threads.

23. The method according to claim 20 further including providing a FIFO buffer between each thread and at least one of the coprocessors.

24. The method according to claim 23 wherein the FIFO buffer is between each thread and the counter coprocessor.

25. The method according to claim 23 wherein the FIFO buffer interface is between each thread and the policy coprocessor.

26. The method of claim 18 including the step of providing specific operating instructions executed by the CLPs which result in commands to control coprocessor operation, which commands flow through the interface between the CLPs and the coprocessors.

27. The method according to claim 26 wherein the operating instructions enable conditional execution of specific coprocessor operation.

28. The method according to claim 27 wherein the execution is either direct or indirect.

29. The method according to claim 18 including the step of providing instructions that enable the system to identify long latency events and short latency events according to the expected response time to a particular

coprocessor command, and to grant full control to another thread when execution of an active thread stalls due to a long latency event, or to grant temporary control to another thread when execution of an active thread stalls due to a short latency event.